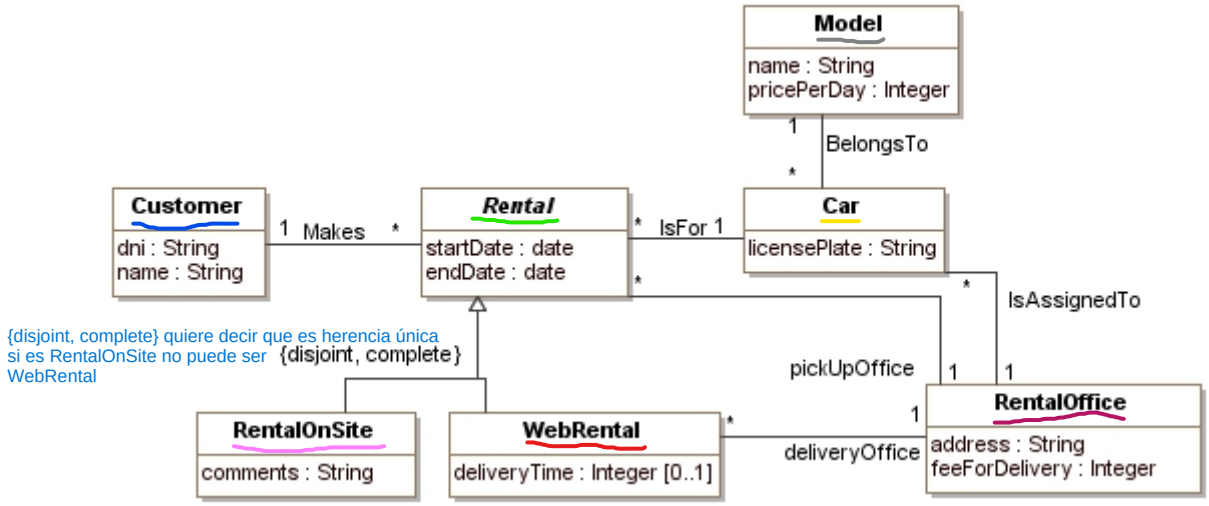
Práctica 4. Alquiler de coches

Una empresa nos ha contratado para desarrollar un software para gestionar los alquileres de coches que hace a sus clientes. Los clientes pueden alquilar coches de diferentes modelos durante un periodo de tiempo. Los coches se tienen que recoger en una oficina de recogida. Los alquileres se pueden hacer directamente en las oficinas de la empresa o bien mediante la aplicación web que la empresa pone a disposición de sus clientes. Para los alquileres que se hacen a través de las oficinas se pueden registrar, si fuera necesario, los comentarios sobre el alquiler. Este tipo de alquileres obliga a los clientes a devolver el coche en la misma oficina donde se ha recogido. Para los alquileres que se hacen vía web se registra la hora de retorno del coche. En estos tipos de alquileres los clientes pueden devolver los coches en una oficina diferente de la donde se ha recogido pagando un cargo por entrega.

Este software es una parte de un sistema más grande. Obviaremos muchos de los elementos que tendría un sistema real. La razón no es otra que hacer una práctica didáctica, y que la dificultad y dedicación prevista sean las deseadas.

Disponemos ya de un análisis previo que podemos utilizar como punto de partida, pero que habrá que corregir y mejorar. A continuación, disponéis del diagrama de clases de esta parte del software:

Diagrama de clases



Restricciones de integridad:

- Un cliente no puede tener alquileres solapados. Solo 1 rental a la vez (igualmente, hay que guardar un historial)
- La fecha de inicio de un alquiler tiene que ser anterior a la fecha final del alquiler.
- La oficina de recogida de un coche de alquiler tiene que ser la misma que la oficina donde está asignado el coche de alquiler. Si el coche está en oficina Joge, se tiene que recoger en oficina Joge
- Si la oficina de recogida y de entrega de un alquiler web son diferentes, la hora de entrega del coche de alquiler tiene que ser anterior a las 13 horas. Aparte de pagar una fee, solo se puede entregar antes de las 13:00

Nuestro sistema tiene que ser capaz de crear alquileres web para los clientes que lo deseen. Para crear estos alquileres es necesario disponer de la información del dni del cliente, la fecha inicial y final del alquiler, la matrícula del coche a alquilar y la oficina de entrega o devolución. Fijaos que la oficina de recogida es la misma que la oficina donde está asignado el coche. Cuando el cliente devuelva el coche se registrará la hora de la devolución.

1. Ejercicio 1

Supongamos ahora que queremos definir una operación numberOfRentalsWithDifferentOffices(): Integer en la clase Customer que devuelve el número de alquileres web que ha hecho un cliente self donde la oficina de recogida y de entrega es diferente. En este momento del diseño del sistema todavía no sabemos qué estructura de datos utilizaremos para guardar los alquileres que ha hecho/hace un cliente. Se pide:

a) ¿Qué patrón de diseño utilizarías para diseñar esta operación? Justifica tu respuesta.
b) Muestra la parte del diagrama de clases que se ve modificada como resultado de la aplicación del patrón utilizado.
c) Muestra el código Java de la operación *numberOfRentalsWithDifferentOffices():Integer* de la clase *Customer*.

2. Ejercicio 2

A menudo, los coches que la empresa de alquiler de coches pone a disposición de sus clientes se tienen que poner fuera de servicio (por reparaciones, para pasar la ITV, etc.). Queremos representar esta situación en nuestro sistema para que cuando los coches estén fuera de servicio no puedan ser alquilados aunque registraremos, si hay, un coche que lo sustituye. Es por eso que nuestro sistema tendrá que proporcionar dos funcionalidades. La funcionalidad (1) poner un coche fuera de servicio (si ya está fuera de servicio o si está en servicio pero es sustituto de algún coche que está fuera de servicio, la funcionalidad no tendrá ningún efecto). Esta funcionalidad pondrá el coche fuera de servicio y registrará la fecha hasta la cual estará fuera de servicio y, si hay, buscará y registrará también un coche sustituto (será cualquier coche del mismo modelo del coche que se pone fuera de servicio que esté asignado a la misma oficina y que esté en servicio). La funcionalidad (2) pone un coche que estaba fuera de servicio en servicio.

En concreto, nos centraremos en implementar la funcionalidad (1) añadiendo la operación *takeOutOfService(backToService:date)* de la clase *Car*. No hace falta implementar la funcionalidad (2).

a) ¿Qué patrón de diseño recomendarías para representar la información descrita (si es que recomiendas alguno)?
b) Muestra el diagrama de clases resultante de añadir estas funcionalidades.
c) Muestra el código Java de la operación *takeOutOfService* de la clase *Car*.

3. Ejercicio 3

Cuando los clientes de la empresa de alquiler de coches alquilan un coche, el sistema que estamos construyendo tiene que proporcionarles el precio del alquiler. Este precio lo calcula la operación *getPrice():Integer* de la siguiente forma: El precio base será el precio del modelo del vehículo por día (*pricePerDay*) * [*endDate* - *startDate*] y 2).

Además, la empresa de alquiler de coches puede añadir al cálculo de precios la posibilidad de hacer promociones que implican descuentos de estos precios. Inicialmente, la empresa ofrecerá dos tipos de promociones: por cantidad y por porcentaje. La promoción por cantidad permitirá decrementar el precio del alquiler en la cantidad indicada en la promoción. La promoción por porcentaje decrementará el precio del alquiler en el porcentaje indicado en la promoción. Las promociones se asignan a los alquileres en el momento de su creación. Evidentemente, es posible que a algunos alquileres no se les aplique ninguna promoción. Las promociones que se asignan a los alquileres son determinadas por una política de la empresa que no impacta al diseño de nuestra operación (impactará a la operación que crea los alquileres). Eso sí, la empresa quiere que mientras no se haga el pago del alquiler, si aparecen nuevas promociones, se apliquen a los alquileres siempre y cuando sean más favorables (no nos tenemos que preocupar tampoco de estos cambios, son gestionados por otras operaciones). Se pide:

a) ¿Qué patrón de diseño recomendarías para este caso? Justificad vuestra respuesta.
b) Muestra el diagrama de clases resultante de la aplicación del patrón.
c) Muestra el nuevo código Java de la operación *getPrice():Integer*.